NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR
Semester 2, AY2014/15

CS1010 – PROGRAMMING METHODOLOGY

April 2015 Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This paper consists of **NINE (9)** questions and comprises **NINE (9)** printed pages.
2. This is an **OPEN BOOK** assessment.
3. Calculators, computer, laptop (inclusive of any computing devices) and electronic dictionaries are not allowed.
4. Answer all questions, and write your answers in the **ANSWER SHEETS** provided.
5. Fill in your Matriculation Number with a pen, clearly on every page of your ANSWER SHEETS.
6. You may use 2B pencil to write your codes.
7. Note that penalty will be given for programs that are unclear or unnecessarily long.
8. You must submit only the ANSWER SHEETS and no other document.

For questions 1 to 6, study the given programs carefully. Then, write out the output of each program in the Answer Sheets. If there is no output (infinite loop etc), then state "No Output" in the Answer Sheets (instead of leaving it blank). We assume there are no compilation errors (warnings are ignored) to all the given programs.

Q1 Given program: (3 Marks)

```
#include <stdio.h>

int main (void) {
   int i = 7;
   int x = 0;

   for ( ; i <= 12; )
   {
      x++;
      switch ( i % 4 )
      {
         case 0:
         case 1: i-2;
         case 2: i++;
                 continue;
         case 3: i++;
      }
      i++;
   }
   printf("x = %d\n", x);
   return 0;
}
```

Q2 Given program: (3 Marks)

```
#include <stdio.h>
#include <string.h>

int main (void) {

  char str1[] = "apple";
  char str2[] = "pie";
  strcpy ( &str1[1], str2);
  printf("str1 = %s, str2 = %s\n", str1, &str2[1]);

  return 0;
}
```

Q3 Given program: (3 Marks)

```
#include <stdio.h>
#include <string.h>
typedef struct {
   char n[20];
   int a;
} _t;

int main (void) {
   _t  s1 = {"mike", 20},
       s2 = {"bob", 18},
       s3 = s1;
   strcpy (s1.n, s2.n);
   strcpy (s2.n, "cat");
   s1.a = s2.n[0] - 'a';
   printf("%s %d \n%s %d\n%s %d \n",
           s1.n, s1.a, s2.n, s2.a, s3.n, s3.a);
   return 0;
}
```

Q4 Given program: (3 Marks)

```
#include <stdio.h>
#define MAX_ENTRIES  8

 int main (void) {
    int entry[MAX_ENTRIES] = {0, 0, 1, 0, 4, 0, 0, 2};
    int i, j, entry2[MAX_ENTRIES];

    for (i=MAX_ENTRIES/2; i > 0; i/=2) {
        for (j=0; j < MAX_ENTRIES; j++)
            if (j-i < 0)
                entry2[j] = entry[j];
            else
                entry2[j] = entry[j] + entry[j - i];

        for ( j=0; j < MAX_ENTRIES; j++)
            entry[j] = entry2[j];
    };
    for (j=0; j < MAX_ENTRIES; j++)
        printf("%d ", entry[j]);

    printf("\n");
    return 0;
 }
```

Q5 Given program: (3 Marks)

```
#include <stdio.h>
int f (int );
int g (int );

int main (void) {
   printf("answer = %d\n", f(7));
   return 0;
}

int f(int y) {
   if (y < 5)
       return g(y);
   else
       return f(y-1) + y;
}

int g(int y) {

   if (y == 0) return 1;

   if (y < 3)
       return f(y-1) + y*y;
   else
       return g(y-1) + 5*y;
}
```

Q6 Given program: (3 Marks)

```
#include <stdio.h>
void f(int, int *);

int main (void) {

   int s = 2147483647; // max value for an int
   int t;

   f(s, &t);
   printf("answer = %d\n", t);
   return 0;
}

void f(int s, int *t) {

   int digit = s%10;

   if (s == 0)
   {
      *t = 0;
      return;
   };

   if (digit == 1 || digit == 3 || digit == 7)
   {
       f(s/10, t);
       *t = *t*10;
   }
   else
   {
       f(s/10, t);
       *t = *t*10 + digit;
   };

}
```

Q7 For the following program, when it is executed till where the arrow indicated (before the executing the return statement), you draw a diagram to show the various variables and their contents in the (computer) memory. For variables that are pointers, please use arrows to point from the variables to the memory locations where the variables have the memory locations in their contents.

(4 Marks)

```
  #include <stdio.h>

  int f(int *, int *, int );

  int main (void) {
    int a = 9, b = -2, c = 5;

    f (&a, &b, c);
    printf("a = %d, b = %d, c = %d\n", a, b, c);
    return 0;
  }

  int f(int *b, int *c, int a) {

    int *d;

    d = b;
    *d = *d + *c;
    b = &a;
    *b = *b + a;
→   return 0;
  }
```

Q8 Class Enrolment

This question refers to Unit 10 in our lecture about the class enrolment. A class enrolment system can be represented by a 2D array **enrol**, where the rows represent the classes, and columns the students. For simplicity, classes and students are identified by non-negative integers. A **1** in **enrol[c][s]** indicates student s is enrolled in class c; a **0** means s is not enrolled in c. With this 2D array, we can ask a few queries.

```
#define MAX_CLASSES 10
#define MAX_STUDENTS 30
int main(void) {
   int enrol[MAX_CLASSES][MAX_STUDENTS] = { {0} }, numClasses, numStudents;

   printf("Number of classes and students: ");
   scanf("%d %d", &numClasses, &numStudents);
   readInputs(enrol, numClasses, numStudents);

   return 0;
}
```

```
3 8
15
3 1
0 0
0 1
1 2
2 0
2 1
2 2
3 2
7 1
6 0
5 0
4 1
4 0
6 2
6 1
```

```
// Read data into array enrol
void readInputs(int enrol[][MAX_STUDENTS],
                int numClasses, int numStudents) {
  int entries;   // number of data entries
  int i, class, student;

  printf("Number of data entries: ");
  scanf("%d", &entries);

  printf("Enter %d data entries (student class): \n", entries);
  // Read data into array enrol
  for (i = 0; i < entries; i++) {
    scanf("%d %d", &student, &class);
    enrol[class][student] = 1;
  }
}
```

	0	1	2	3	4	5	6	7
0	1	0	1	0	1	1	1	0
1	1	0	1	1	1	0	1	1
2	0	1	1	1	0	0	1	0

Query: Name all students who are enrolled in all classes

```
// Find students who are enrolled in all classes
void busiestStudents(int enrol[][MAX_STUDENTS],
                     int numClasses, int numStudents) {
  int sum;
  int r, c;

  printf("Students who take all classes: ");
  for (c = 0; c < numStudents; c++) {
    sum = 0;
    for (r = 0; r < numClasses; r++) {
      sum += enrol[r][c];
    }
    if (sum == numClasses)
      printf("%d ", c);
  }
  printf("\n");
}
```

	0	1	2	3	4	5	6	7
0	1	0	1	0	1	1	1	0
1	1	0	1	1	1	0	1	1
2	0	1	1	1	0	0	1	0
	2	1	3	2	2	1	3	1

Column sums

Query: Name any class with the most number of students

```
int classWithMostStudents
      (int enrol[][MAX_STUDENTS],
       int numClasses, int numStudents) {
  int classSizes[MAX_CLASSES];
  int r, c; // row and column indices
  int maxClass, i;

  for (r = 0; r < numClasses; r++)
    classSizes[r] = 0;
    for (c = 0; c < numStudents; c++) {
      classSizes[r] += enrol[r][c];
  }

  // find the one with most students
  maxClass = 0;  // assume class 0 has most students
  for (i = 1; i < numClasses; i++)
    if (classSizes[i] > classSizes[maxClass])
      maxClass = i;

  return maxClass;
}
```

	0	1	2	3	4	5	6	7	*Row sums*
0	1	0	1	0	1	1	1	0	5
1	1	0	1	1	1	0	1	1	6
2	0	1	1	1	0	0	1	0	4

You are to write two functions as follows:

(A) function **secondMostNumOfCourses** to find out the second most number of courses taken by students. For example, with 10 students each taking 4, 5, 3, 5, 6, 5, 4, 6, 3, and 4 courses (note that this is not the same example as shown in the above screen shots), the most number of courses taken by students is 6, and the second most number of courses taken by students is 5 (which is the answer needed here). We assume that not all the students taking the same number of courses.

(5 Marks)

(B) function **mostPopularNumOfCourses** to find out the most popular number(s) of courses taken by students. Consider the same example as in Part (A) where there are 10 students each taking 4, 5, 3, 5, 6, 5, 4, 6, 3, and 4 courses. Then, we have 3 students taking 4 courses and 3 other students taking 5 courses, while there are fewer than 3 students taking other numbers of courses. So, the answers needed here are 4 and 5.

(5 Marks)

Q9 Parking Simulation

This simulation is to find out what is the utilization rate of a roadside available for parking (without drawing explicit parking lots on the roadside, but we do assume the roadside is discretized in some way still). Here are the details:

1. We discretize the roadside into MAX_PARKING_UNIT (say 9000) of **parkingUnits**. This is to say that a car can be parked at location 0, 1, 2, ...till (MAX_PARKING_UNIT – LENGTH_OF_CAR) of **parkingUnits**.

2. A car has length equal to LENGTH_OF_CAR units (say 8). This is to say that if a car is to park at location **x**, then it will occupy the roadside **parkingUnits** from **x** till (**x** + LENGTH_OF_CAR – 1). In other words, for the car to decide to park at location **x**, the roadside **parkingUnits** from **x** till (**x** + LENGTH_OF_CAR – 1) must be unoccupied by other car at that time.

3. During each round of the simulation, a random number **x** between 0 till (MAX_PARKING_UNIT – 1) is first generated. With **x**, the program is to search from location **x** forward (meaning **x**, then **x**+1, etc.) for the first contiguous LENGTH_OF_CAR units of unoccupied **parkingUnits** to park the car. (For simplicity here, we do not search "backward" from location **x**.) If the search till the location MAX_PARKING_UNIT–1 (or even earlier if you can tell) is unsuccessful, this round is ended with not parking the car; else this round succeeds in parking the car, and needs to mark those **parkingUnits** found as occupied by the car being parked.

4. The number of round for the simulation is set as MAX_NUM_ROUND (say 10000). Having completed the MAX_NUM_ROUND, the program calculates the total number of **parkingUnits** occupied by cars, and divides this number by MAX_PARKING_UNIT to print out the utilization rate of the roadside for parking (in % with float of 2 digits after the decimal point).

5. We do not consider car left after parking for some time.

NOTE: if you do try out this simulation in your computer with different values of LENGTH_OF_CAR and MAX_PARKING_UNIT, you will observe that the utilization rate is actually quite consistent.

(8 Marks)

=== END OF PAPER ===